Assessing the cost of redistribution followed by a computational kernel: Complexity and performance results

Julien Herrmann, George Bosilca, Thomas Hérault, Loris Marchal, Yves Robert, Jack Dongarra

Assessing the cost of redistribution followed by a computational kernel: complexity and performance results

Julien Herrmann[1], George Bosilca[2], Thomas Hérault[2],
Loris Marchal[1], Yves Robert[1,2] and Jack Dongarra[2]

1. Ecole Normale Supérieure de Lyon, CNRS & INRIA, France
{julien.herrmann|loris.marchal|yves.robert}@ens-lyon.fr
2. University of Tennessee Knoxville, USA
{bosilca|herault|dongarra}@icl.utk.edu

April 25, 2015

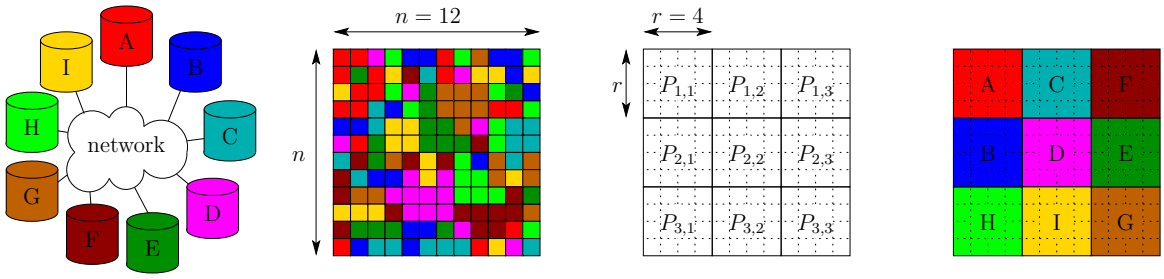
Abstract

The classical redistribution problem aims at optimally scheduling communications when reshuffling from an initial data distribution to a target data distribution. This target data distribution is usually chosen to optimize some objective for the algorithmic kernel under study (good computational balance or low communication volume or cost), and therefore to provide high efficiency for that kernel. However, the choice of a distribution minimizing the target objective is not unique. This leads to generalizing the redistribution problem as follows: find a re-mapping of data items onto processors such that the data redistribution cost is minimal, and the operation remains as efficient. This paper studies the complexity of this generalized problem. We compute optimal solutions and evaluate, through simulations, their gain over classical redistribution. We also show the NP-hardness of the problem to find the optimal data partition and processor permutation (defined by new subsets) that minimize the cost of redistribution followed by a simple computational kernel. Finally, experimental validation of the new redistribution algorithms are conducted on a multicore cluster, for both a 1D-stencil kernel and a more compute-intensive dense linear algebra routine.

1 Introduction

In parallel computing systems, data locality has a strong impact on application performance. To achieve good locality, a redistribution of the data may be needed between two different phases of the application, or even at the beginning of the execution, if the initial data layout is not suitable for performance. Data redistribution algorithms are critical to many applications, and therefore have received considerable attention. The data redistribution problem can be stated informally as follows: given N data items that are currently distributed across P processors, redistribute them according to a different target layout. Consider for instance a dense square matrix $A = (a_{ij})_{0 \leq i,j < n}$ of size n, whose initial distribution is random, and that must be redistributed into square blocks across a $p \times p$ 2D-grid layout. A scenario for this problem is that the matrix has been generated by a Monte-Carlo method and is now needed for some matrix product $C \leftarrow C + AB$. Assume for simplicity that p divides n, and let $r = n/p$. In this example, $N = n^2$, $P = p^2$, and the redistribution will gather a block of $r \times r$ data elements of A on each processor, as illustrated on Figure 1. More precisely, all the elements of block $A_{i,j} = (a_{k,\ell})$, where $ri \leq k < (r+1)i$ and $rj \leq \ell < (r+1)j$, must be sent to processor $P_{i,j}$. This example illustrates the classical redistribution problem. Depending upon the cost model for communications, various optimization objectives have been considered, such as the total volume of data that is moved from one processor to another, or the total time for the redistribution, if several communications can take place simultaneously. We detail classical cost models in Section 2, which is devoted to related work.

Modern computing platforms are equipped with interconnection switches and routing mechanisms mapping the most usual interconnection graphs onto the physical network with reduced (or even negligible) dilation and contention. Continuing with the example, the $p \times p$ 2D-grid will be virtual, i.e.,



(a) processors holding the data (b) intial data distribution (c) target data partition (d) final data distribution

Figure 1: Example of matrix redistribution with $N = 12^2$ data blocks and $P = 3^2$ processors. Each color in the data distributions corresponds to a processor, e.g., all red data items reside on processor A.

an overlay topology mapped into the physical topology, forcing the interconnection switch to emulate a 2D-grid. Notwithstanding, the layout of the processors in the grid remains completely flexible. For instance, the processors labeled $P_{1,1}$, $P_{1,2}$ and $P_{2,1}$ can be *any* processors in the platform, and we have the freedom to choose which three processors will indeed be labeled as the top-left corner processors of the virtual grid. Now, to describe the matrix product on the 2D-grid, we say that data will be sent *horizontally* between $P_{1,1}$ and $P_{1,2}$, and vertically between $P_{1,1}$ and $P_{2,1}$, but this actually means that these messages will be routed by the actual network, regardless of the physical position of the three processors in the platform.

This leads us to revisit the redistribution problem, adding the flexibility to select the *best* assignment of data on the processors (according to the cost model). The problem can be formulated as mapping a *partition* of the initial data onto the resources: there are P data subsets (the blocks in the example) to be assembled onto P processors, with a huge (exponential) number, namely $P!$, of possible mappings. An intuitive view of the problem is to assign the same color to all data items that initially reside on the same processor, and to look for a coloring of the virtual grid that will minimize the redistribution cost. For instance, in Figure 1, most data items of the block allocated to the virtual processor $P_{1,1}$ are initially colored red (they reside on the red processor A), so we decided to map $P_{1,1}$ on processor A to avoid moving these items.

One major goal of this paper is to assess the complexity of the problem of finding the best processor mapping for a given initial data distribution and a target data partition. This amounts to determining the processor assignment that minimizes the cost of redistributing the data according to the partition. There are $P!$ possible redistributions, and we aim at finding the one minimizing a predefined cost-function. In this paper, we use the two most widely-used criteria in the literature to compute the cost of a redistribution:

- **Total volume.** In this model, the platform is not dedicated, and the objective is to minimize the total communication volume, i.e., the total amount of data sent from one processor to another. Minimizing this volume makes it less likely to disrupt the other applications running on the platform, and is expected to decrease network contention,, hence redistribution time. Conceptually, this is equivalent to assuming that the network is a bus, globally shared by all computing resources.

- **Number of parallel steps.** In this model, the platform is dedicated to the application, and several communications can take place in parallel, provided that they involve different processor pairs. This is the one-port bi-directional model used in [1, 2]. The quantity to minimize is the number of parallel steps, where a step is a collection of unit-size messages that involve different processor pairs.

One major contribution of this paper is the design of an algorithm solving this optimization problem for either criterion. We also provide various experiments to quantify the gain that results from choosing the optimal mapping rather than a *canonical* mapping where processors are labeled arbitrarily, and independently of the initial data distribution.

As mentioned earlier, a redistribution is usually motivated by the need to efficiently execute in parallel a subsequent computational kernel. In most cases, there may well be several data partitions that are suitable for the efficient execution of this kernel. The optimal partition also depends upon the initial

data distribution. Coming back to the introductory example, where the redistribution is followed by a matrix product, we may ask whether a full block partition is absolutely needed? If the original data is distributed along a suitable, well-balanced distribution, a simple solution is to compute the product in place, using the *owner computes* rule, that is, we let the processor holding $C_{i,j}$ compute all $A_{i,k}B_{k,j}$ products. This means that elements of A and B will be communicated during the computation, when needed. On the contrary, if the original distribution has a severe imbalance, with some processors holding significantly more data than others, a redistribution is very likely needed. But in this latter case, do we really need a perfect full block partition? In fact, the optimization problem is the following: given an initial data distribution, what is the best data partition, and the best mapping of this partition onto the processors, to minimize total execution time, defined as the sum of the redistribution time and of the execution of the kernel. Another major contribution of this paper is to assess the complexity of this intricate problem. Finding the optimal partition mapping becomes NP-complete when coupling the redistribution with a simple computational kernel such as an iterative 1D-stencil kernel. Here the optimization objective is the sum of the redistribution time (computed using either of the two criteria above, with all communications serialized or with communications organized in parallel steps), and of the parallel execution time of a few steps of the stencil. Intuitively this confirms that determining the optimal data partition and its mapping is a difficult task. Stencil computations naturally favor block distributions, in order to communicate only block frontiers at each iteration. But this has to be traded-off with the cost of moving the data from the initial distribution, with the number of iterations, and with the possible imbalance of the final distribution that is chosen (whose own impact depends upon the communication-to-computation ratio of the machine). Altogether, it is no surprise that all these possibilities lead to a hard combinatorial problem.

Finally, this paper provides an experimental validation of the new redistribution algorithms conducted on a multicore cluster. We first experiment with the 1D-stencil algorithm and obtain performance improvements in total execution time that strongly depend on initial distributions. Different data configurations have been tested to assess this gain. For a more compute-intensive dense linear algebra routine, such as QR factorization, redistributing the data items can also be necessary. The 2D block-cyclic partition is known to offer a good trade-off between the amount of communications during the QR factorization and the load balancing among processors. Using the algorithms to determine the best distribution compatible with the 2D block-cyclic partition provides significant improvement in the completion time.

The rest of the paper is organized as follows. We survey related work in Section 2. We detail the model and formally state the optimization problems in Section 3. We deal with the problem of finding the best redistribution for a given data partition in Section 4. Sections 4.1 and 4.2 provide algorithms computing the optimal solution, while Section 4.3 reports simulation results showing the gain over redistributing to an arbitrary compatible distribution. In Section 5, we couple the redistribution with a stencil kernel, and show that finding the optimal data partition, together with the corresponding redistribution, is NP-complete. Experiments conducted on a multicore cluster are reported in Section 6. Section 6.1 is devoted to the experimental setup. Section 6.2 provides results when redistribution is followed by a stencil kernel, while Section 6.3 deals with QR factorization. We provide final remarks and directions for future work in Section 7.

2 Related work

2.1 Communication model

The *macro-dataflow model* has been widely used in the scheduling literature (see the survey papers [3, 4, 5, 6] and the references therein). In this model, the cost to communicate L bytes is $\alpha + L\beta$, where α is a start-up cost and β is the inverse of the bandwidth. In this paper, we consider large, same-sized data items, so we can safely restrict to *unit* communications that involve a single data item; we integrate the start-up cost into the cost of a unit communication.

In the macro-dataflow model, communication delays from one task to its successor are taken into account, but communication resources are not limited. First, a processor can send (or receive) any number of messages in parallel, hence an unlimited number of communication ports is assumed (this explains the name *macro-dataflow* for the model). Second, the number of messages that can simultaneously circulate

between processors is not bounded, hence an unlimited number of communications can simultaneously occur on a given link. In other words, the communication network is assumed to be contention-free, which of course is not realistic as soon as the processor number exceeds a few units.

A much more realistic communication model is the *one-port bidirectional model* where at a given time-step, any processor can communicate with at most one other processor in both directions: sending to and receiving from. Thus, communications can occur in parallel, provided that they involve disjoint pairs of sending/receiving processors. The one-port model was introduced by Hollermann et al. [1], and Hsu et al. [2]. It has been widely used since, both for homogeneous and heterogeneous platforms [7, 8].

2.2 General data redistribution

The complexity of scheduling data redistribution in distributed architectures strongly depends on the network model. When the network has a general graph topology, achieving the minimal completion time for a set of communications is NP-complete, even when the time required to move a data along any link is constant [9].

In this context, several variants of the *one-port bidirectional model* have been considered. The first variant is an unidirectional one-port model, where a processor can participate in only one communication at a time (either as a sender or as a receiver); with this variant, the redistribution problem becomes NP-complete [10]. A second variant consists of assuming that each processor p has a number of ports $v(p)$ that represents the maximum number of simultaneous transfers that this processor can be involved in [11]. Finally, in a third variant [12], processors have memory constraints that must be enforced during the redistribution process.

2.3 Array redistribution

A specific class of redistribution problems has received considerable attention, namely the redistribution of arrays that are already distributed in a block-cyclic fashion over a multidimensional processor grid. This interest was originally motivated by the HPF [13] programming style, in which scientific applications are decomposed into phases. At each phase, there is an optimal distribution of the data arrays onto the processor grid. Typically, arrays are distributed according to a `CYCLIC(r)` pattern[1] along one or several dimensions of the grid. The best value of the distribution parameter r depends on the characteristics of the algorithmic kernel as well as on the communication-to-computation ratio of the target machine [14]. Because the optimal value of r changes from phase to phase and from one machine to another (think of a heterogeneous environment), run-time redistribution turns out to be a critical operation, as stated in [15, 16, 17, 18] (among others). Communications are scheduled into parallel steps, which involve different processor pairs. The model comes in two variants, synchronous and asynchronous. In the synchronous variant, the cost of a parallel step is the maximal size of a message and the objective is to minimize the sum of the costs of the steps [16, 19]. In the asynchronous model, some overlap is allowed between communication steps [20]. Finally, the ScaLAPACK library provides a set of routines to perform array redistribution [21]. A total exchange is organized between processors, which are arranged as a (virtual) caterpillar. The total exchange is implemented as a succession of synchronous steps.

We point out that all the works referenced in Section 2.2 and in this one deal with a fixed target distribution. To the best of our knowledge, this work is the first to consider target data partitions rather than target data distributions, thereby allowing to choose the best data redistribution among $P!$ candidates, where P is the number of enrolled processors. Also, this work is the first to study the cost of coupling a redistribution with a computational kernel, which is a very important problem in practice.

3 Model and framework

This section details the framework and formally states the optimization problems. We start with a few definitions.

[1] The definition is the following: let an array $X[0...M-1]$ be distributed according to a block-cyclic distribution `CYCLIC(r)` onto a linear grid of P processors. Then element $X[i]$ is mapped onto processor $p = \lfloor i/r \rfloor \bmod P$, $0 \leq p \leq P-1$.

3.1 Definitions

Consider a set of N data items (numbered from 0 to $N-1$) distributed onto P processors (numbered from 0 to $P-1$).

Definition 1 (Data distribution)**.** A *data distribution* $\mathcal{D}$ defines the mapping of the elements onto the processors: for each data item x, $\mathcal{D}(x)$ is the processor holding it.

Definition 2 (Data partition)**.** A *data partition* $\mathcal{P}$ associates to each data item x an index $\mathcal{P}(x)$ ($0 \leq \mathcal{P}(x) \leq P-1$) so that two data items with the same index reside on the same processor (not necessarily processor $\mathcal{P}(x)$). The j^{th} *component* of the data partition $\mathcal{P}$ is the subset of the data items x such that $\mathcal{P}(x) = j$.

It is straightforward to see that a data distribution $\mathcal{D}$ defines a single corresponding data partition $\mathcal{P} = \mathcal{D}$. However, a given data partition does not define a unique data distribution. On the contrary, any of the $P!$ permutations of $\{0, \ldots, P-1\}$ can be used to map a data partition onto the processors.

Definition 3 (Compatible distribution)**.** A data distribution $\mathcal{D}$ is *compatible* with a data partition $\mathcal{P}$ if and only if there exists a permutation of processors σ of $\{0, \ldots, P-1\}$ such that for each data item x, $\mathcal{D}(x) = \sigma(\mathcal{P}(x))$.

3.2 Cost of a redistribution

In this section, we formally state the two metrics for the cost of a redistribution, namely the total volume and the number of parallel steps. Both metrics assume that the communication of one data item from one processor to another takes the same amount of time, regardless of the item and of the location of the source and target processors. Indeed, data items can be anything from single elements to matrix tiles, columns or rows, so that our approach is agnostic of the granularity of the redistribution. As already mentioned, many modern interconnection networks are fully-connected switches, and they can implement any (same-length) communication in the same amount of time. Note that with asymmetric networks, it is always possible to use the worst-case communication time between any processor pair as the unit time for a communication.

3.2.1 Total volume

For this metric, we simply count the number of data items that are sent from one processor to another. This metric may be pessimistic if some parallelism is possible, but it provides an interesting measure of the overhead of the redistribution, especially if the platform is not dedicated.

Given an initial data distribution $\mathcal{D}_{ini}$ and a target distribution $\mathcal{D}_{tar}$, for $0 \leq i, j \leq P-1$, let $q_{i,j}$ be the number of data items that processor i must send to processor j: $q_{i,j}$ is the number of data items x such that $\mathcal{D}_{ini}(x) = i$ and $\mathcal{D}_{tar}(x) = j$. For a given processor i, let s_i (respectively r_i) be the total number of data items that processor i must send (respectively receive) during the redistribution. We have $s_i = \sum_{j \neq i} q_{i,j}$ and $r_i = \sum_{j \neq i} q_{j,i}$. The total communication volume of the redistribution is defined as $\mathit{RedistVol}(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = \sum_i s_i = \sum_i r_i$.

3.2.2 Number of parallel steps

With this metric, some communications can take place in parallel, provided that each of them involves a different processor pair (sender and receiver). This communication model is the bidirectional one-port model introduced in [1, 2] and accounts for contention when communications take place simultaneously.

We define a parallel step as a set of unit-size communications (one data item each) such that all senders are different, and all receivers are different (the set of senders of the set of receivers are not necessary disjoint). With this definition, a processor can send and receive a data item at the same time but can not send (respectively receive) a data item to (respectively from) more than one processor during the same communication step. Given an initial data distribution $\mathcal{D}_{ini}$ and a target distribution $\mathcal{D}_{tar}$, we define $\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar})$ as the minimal number of parallel steps that are needed to perform the redistribution.

3.3 Optimization problems

Here, we formally introduce the optimization problems that we study in Sections 4 and 5 below.

3.3.1 Best redistribution compatible with a given partition

In the optimization problems of Section 4, the data partition is given, and we aim at finding the best compatible target distribution (among $P!$ ones). More precisely, given an initial data distribution $\mathcal{D}_{ini}$ and a target data partition $\mathcal{P}_{tar}$, we aim at finding a data distribution $\mathcal{D}_{tar}$ that is compatible with $\mathcal{P}_{tar}$ and such that the redistribution cost from $\mathcal{D}_{ini}$ to $\mathcal{D}_{tar}$ is minimal. Since we have two cost metrics, we define two problems:

Definition 4 (VolumeRedistrib)**.** Given $\mathcal{D}_{ini}$ and $\mathcal{P}_{tar}$, find $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$ such that $\mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ is minimized.

Definition 5 (StepRedistrib)**.** Given $\mathcal{D}_{ini}$ and $\mathcal{P}_{tar}$, find $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$ such that $\mathit{RedistSteps}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ is minimized.

We show in Section 4 that both problems have polynomial complexity.

3.3.2 Best partition and best compatible redistribution

In the optimization problems of Section 5, the data partition is no longer fixed. Given an initial data distribution $\mathcal{D}_{ini}$, we aim at executing some computational kernel whose cost $T_{comp}(\mathcal{P}_{tar})$ depends upon the data partition $\mathcal{P}_{tar}$ that will be selected. Note that this computational kernel will have the same execution cost for any distribution $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$, because of the symmetry of the target platform. However, the redistribution cost from $\mathcal{D}_{ini}$ to $\mathcal{D}_{tar}$ will itself depend upon $\mathcal{D}_{tar}$. We model the total cost as the sum of the time of the redistribution and of the computation. Letting τ_{comm} denote the time to perform a communication, the time to execute the redistribution is either $\mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm}$ or $\mathit{RedistSteps}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm}$, depending upon the communication model. This leads to the following two problems:

Definition 6 (VolPart&Redistrib)**.** Given $\mathcal{D}_{ini}$, find $\mathcal{P}_{tar}$, and $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$, such that $T_{total} = \mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm} + T_{comp}(\mathcal{P}_{tar})$ is minimized.

Definition 7 (StepPart&Redistrib)**.** Given $\mathcal{D}_{ini}$, find $\mathcal{P}_{tar}$, and $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$, such that $T_{total} = \mathit{RedistSteps}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm} + T_{comp}(\mathcal{P}_{tar})$ is minimized.

Note that both problems require that we are able to compute $T_{comp}(\mathcal{P}_{tar})$ for any target data partition $\mathcal{P}_{tar}$. This is realistic only for very simple computational kernels. In Section 5, we consider such a kernel, namely the 1D-stencil. We show the NP-completeness of both VolPart&Redistrib and StepPart&Redistrib for this kernel, thereby assessing the difficulty to couple redistribution and computations.

4 Redistribution

This section deals with the VolumeRedistrib and StepRedistrib problems: given a data partition $\mathcal{P}_{tar}$ and an initial data distribution $\mathcal{D}_{ini}$, find one target distribution $\mathcal{D}_{tar}$ among all possible $P!$ compatible target distributions that minimizes the cost of the redistribution, either expressed in total volume or number of parallel steps. We show that both problems have polynomial complexity. As a side note, we point out that these results directly extend to the case where we have different numbers of processors for the source and target distributions.

4.1 Total communication volume

Theorem 1. *Given an initial data distribution $\mathcal{D}_{ini}$ and target data partition $\mathcal{P}_{tar}$, Algorithm 1 computes a data distribution $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$ such that $\mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar})$ is minimized, and its complexity is $O(NP^2 + P^3)$.*

Proof. Using the definition of s_i and r_i from Section 3.2.1, the total volume of communications during the redistribution phase from the initial distribution to the target distribution is

$$\mathit{RedistVol}(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = \sum_{0 \leq i \leq P-1} s_i = \sum_{0 \leq i \leq P-1} r_i.$$

Solving VolumeRedistrib amounts to finding a one-to-one perfect matching between each component of the target data partition and the processors, so that the total volume of communications is minimized. Algorithm 1 builds the complete bipartite graph where the two sets of vertices represents the P processors and the P components of the target data partition. Each edge (i, j) of this graph is weighted with the amount of data that processor P_i would have to receive if matched to component j of the data partition.

Computing the weight of the edges can be done with complexity $O(NP^2)$. The complexity of finding a minimum-weight perfect matching in a bipartite graph with n vertices and m edges is $O(n(m + n \log n))$ (see Corollary 17.4a in [22]). Here n=P and m=P^2, hence the overall complexity of Algorithm 1 is $O(NP^2 + P^3)$. □

Note that, in Algorithm 1, the complexity of computing egde weights may easily be reduced to $O(NP + P^2)$: (i) we first initialize all weights to 0 (in $O(P^2)$), (ii) then, for each data item x and each $i \neq \mathcal{D}_{ini}(x)$, the weight of edge $(i, \mathcal{P}_{tar}(x))$ is incremented (in $O(NP)$). With this optimization, the complexity of Algorithm 1 can be reduced to $O(NP + P^3)$.

Algorithm 1: BestDistribForVolume

Data: Initial data distribution $\mathcal{D}_{ini}$ and target data partition $\mathcal{P}_{tar}$
Result: a data distribution $\mathcal{D}_{tar}$ compatible with the given data partition, such that $\mathit{RedistVol}(\mathcal{D}_{ini} \to \mathcal{D}_{tar})$ is minimized
$A \leftarrow \{0, \ldots, P-1\}$ (set of processors)
$B \leftarrow \{0, \ldots, P-1\}$ (set of data partition indices)
$G \leftarrow$ complete bipartite graph (V, E) where $V = A \cup B$
for *edge* (i, j) *in* E **do**
  $\mathit{weight}(i, j) \leftarrow |\{x \text{ s.t. } \mathcal{P}_{tar}(x) = j \text{ and } \mathcal{D}_{ini}(x) \neq i\}|$
$\mathcal{M} \leftarrow$ minimum-weight perfect matching of G
for $(i, j) \in \mathcal{M}$ **do**
  for x s.t. $\mathcal{P}_{tar}(x) = j$ **do** $\mathcal{D}_{tar}(x) \leftarrow i$
return $\mathcal{D}_{tar}$

4.2 Number of parallel communication steps

The second metric is the number of parallel communications steps in the bidirectional one-port model. Note that this objective is quite different from the total communication volume: consider for instance a processor which has to send and/or receive much more data than the others; all the communications involving this processor will have to be performed sequentially, creating a bottleneck.

Theorem 2. *Given an initial data distribution $\mathcal{D}_{ini}$ and target data partition $\mathcal{P}_{tar}$, Algorithm 2 computes a data distribution $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$ such that $\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar})$ is minimized, and its complexity is $O(NP^2 + P^{\frac{9}{2}})$.*

Proof. First, given an initial data distribution $\mathcal{D}_{ini}$ and a target distribution $\mathcal{D}_{tar}$, we can compute $\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar})$ as

$$\mathit{RedistSteps}(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = \max_{0 \leq i \leq P-1} max(s_i, r_i).$$

This well-known result [19] is a direct consequence of König's theorem (see Theorem 20.1 in [22]) stating that the edge-coloring number of a bipartite multigraph is equal to its maximum degree.

Algorithm 2 builds the complete bipartite graph G where the two sets of vertices represent the P processors and the P components of $\mathcal{P}_{tar}$. Each edge (i, j) of the complete bipartite graph is weighted

with the maximum between the amount $r_{i,j}$ of data that processor i would have to receive if matched to component j of the data partition, and the amount of data that it would have to send in the same scenario. A one-to-one matching between the two sets of vertices whose maximal edge weight is minimal represents an optimal solution to StepRedistrib. We denote by $\mathcal{M}_{opt}$ such a matching and m_{opt} its maximal edge weight. Since there are P processors and P components in $\mathcal{P}_{tar}$, the one-to-one matching $\mathcal{M}_{opt}$ is a matching of size P.

Algorithm 2 prunes an edge with maximum weight from G until it is not possible to find a matching of size P, and it returns the last matching of size P. We denote by $\mathcal{M}_{ret}$ this matching and m_{ret} its maximum edge weight. Using a proof by contradiction, we first assume that $m_{ret} > m_{opt}$. Then matching $\mathcal{M}_{opt}$ only contains edges with weight strictly smaller than m_{ret}. Since Algorithm 2 prunes edges starting from the heaviest ones, these edges are still in G when Algorithm 2 returns $\mathcal{M}_{ret}$. Thus we can remove the edges with maximal weight m_{ret} in $\mathcal{M}_{ret}$ and still have a matching of size P. This contradicts the stopping condition of Algorithm 2. Thus $m_{ret} = m_{opt}$ and the matching returned by Algorithm 2 is a solution to StepRedistrib.

Again, computing edge weights can be done with complexity $O(NP^2 + P^2)$. Algorithm 2 uses the Hopcroft–Karp Algorithm [23] to find the maximum cardinality matching of a bipartite graph $G = (V, E)$ in time $O(|E|\sqrt{|V|})$. There are no more than P^2 iterations in the while loop, and Algorithm 2 has a worst-case complexity of $O(NP^2 + P^{\frac{9}{2}})$. □

Note that, like in previous section, the complexity of Algorithm 2 can easily be reduced to $O(NP+P^{\frac{9}{2}})$ with an optimized weight computation.

Algorithm 2: BestDistribForSteps

Data: Initial data distribution $\mathcal{D}_{ini}$ and target data partition $\mathcal{P}_{tar}$
Result: A data distribution $\mathcal{D}_{tar}$ compatible with the given data partition so that *RedistSteps*$(\mathcal{D}_{ini} \to \mathcal{D}_{tar})$ is minimized

$A \leftarrow \{0, \ldots, P-1\}$ (set of processors)
$B \leftarrow \{0, \ldots, P-1\}$ (set of data partition indices)
$G \leftarrow$ complete bipartite graph (V, E) where $V = A \cup B$
for *edge* (i, j) *in* E **do**
  $r_{i,j} \leftarrow |\{x \text{ s.t. } \mathcal{P}_{tar}(x) = j \text{ and } \mathcal{D}_{ini}(x) \neq i\}|$
  $s_{i,j} \leftarrow |\{x \text{ s.t. } \mathcal{P}_{tar}(x) \neq j \text{ and } \mathcal{D}_{ini}(x) = i\}|$
  $weight(i, j) \leftarrow \max(r_{i,j}, s_{i,j})$
$\mathcal{M} \leftarrow$ maximum cardinality matching of G (using the Hopcroft–Karp Algorithm)
while $|\mathcal{M}| = P$ **do**
  $\mathcal{M}_{save} \leftarrow \mathcal{M}$
  Suppress all edges of G with maximum weight
  $\mathcal{M} \leftarrow$ maximum cardinality matching of G (using the Hopcroft–Karp Algorithm)
return $\mathcal{M}_{save}$

4.3 Evaluation of optimal vs. arbitrary redistributions

In this section, we conduct several simulations to illustrate the interest of the two algorithms introduced above. In particular, we show that in many cases, it is important to optimize the mapping rather than resorting to an arbitrary mapping which could induce many more communications. Source code for the algorithms and simulations is publicly available at `http://perso.ens-lyon.fr/julien.herrmann/`.

4.3.1 Random balanced initial data distribution

First we consider a random balanced initial data distribution $\mathcal{D}_{ini}$, where each processor initially hosts D data items, and each data item has the same probability to reside on any processor. Most parallel applications require perfect load balancing to achieve good performance, and thus a balanced data partition. Therefore, we consider here a balanced target data partition $\mathcal{P}_{tar}$ (each of its P components

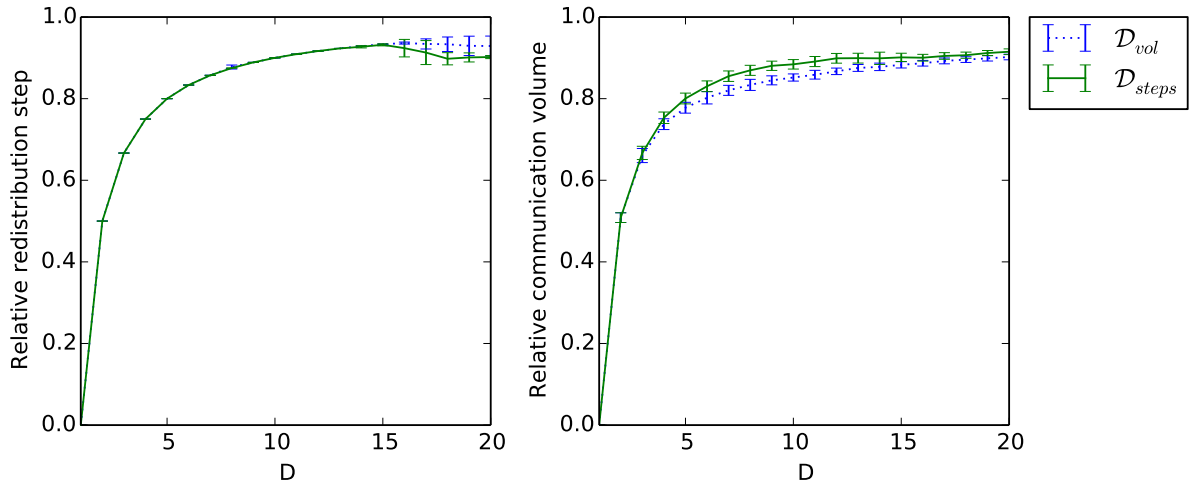


Figure 2: Performance of $\mathcal{D}_{vol}$ (computed by Algorithm 1) and $\mathcal{D}_{steps}$ (computed by Algorithm 2) relatively to the canonical distribution, for a random initial distribution and for both metrics.

includes D data items). We denote by $\mathcal{D}_{can}$ the canonical data distribution (compatible with partition $\mathcal{P}_{tar}$) which maps its j^{th} component onto processor j.

As seen in Section 3, the volume of communication involved during the redistribution from $\mathcal{D}_{ini}$ to $\mathcal{D}_{can}$ is $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{can}) = \sum_{0 \leq j \leq P-1} |\{x \text{ s.t. } \mathcal{P}_{tar}(x) = j \text{ and } \mathcal{D}_{ini}(x) \neq j\}|$. Since each component of $\mathcal{P}_{tar}$ has cardinal D and $\mathcal{D}_{ini}(x)$ is equal to j with a probability $\frac{1}{P}$ for any processor j and any data item x, we can compute the expected volume of communication: $E(RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{can})) = D(P-1)$. Thus, picking an arbitrary target distribution leads to an average volume of communications linear in P.

Each processor hosts D data items at the beginning and at the end of the redistribution phase. Thus, according to Section 4.2, the number of steps required to schedule the redistribution phase is equal to D if and only if one of the P processors has to send its complete initial data set during the redistribution phase. This happens with probability

$$p = 1 - \left(1 - \left(\frac{P-1}{P}\right)^D\right)^P.$$

This probability is equal to 0.986 for $P = 10$ and $D = 10$, and is non-decreasing with P, which means that the worst number of steps is reached in almost all cases for average values of D. This shows that picking an arbitrary data distribution $\mathcal{D}_{can}$ is suboptimal most of the time. Instead, we can use Algorithm 1 to find the data distribution $\mathcal{D}_{vol}$ that minimizes the volume of communications involved in the redistribution phase, and Algorithm 2 to find the data distribution $\mathcal{D}_{steps}$ that minimizes the number of steps of the redistribution phase. Figure 2 depicts the relative volume of communications and the relative number of redistribution steps when using target data distributions $\mathcal{D}_{vol}$ and $\mathcal{D}_{steps}$. The results are normalized with the performance of the arbitrary target distribution $\mathcal{D}_{can}$. The simulations have been conducted with $P = 32$ processors and up to $D = 20$ data items residing on each of them. These values correspond to an application dealing with $32 \times 20 = 640$ data items and running on a distributed cluster of 32 processors, which is a realistic problem size when considering linear algebra problems, since each data item is a matrix tile. For these values, the arbitrary target distribution $\mathcal{D}_{can}$ requires on average 620 communications and involves 20 parallel steps with a probability larger than $1 - 3.3 \times 10^{-11}$. Each point in Figure 2 represents the average results and the standard deviation on a set of 50 random initial distributions. The best data distributions for the communication volume and for the communication steps represent a 10% improvement compared to an arbitrary target distribution when $D \geq 10$, and a larger improvement for smaller values of D. The results for these two data distributions are really close and present a small standard deviation.

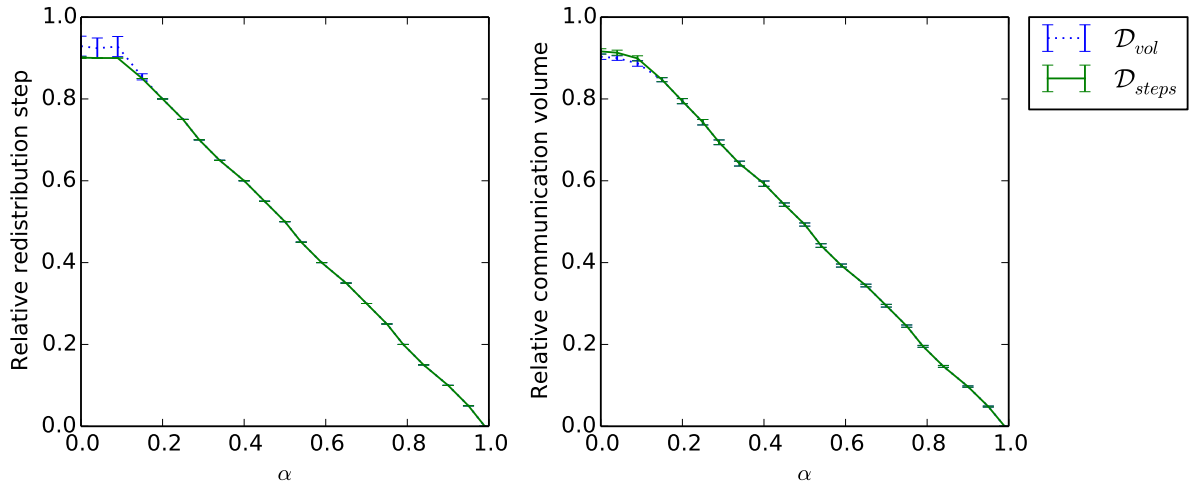


Figure 3: Performance of $\mathcal{D}_{vol}$ (computed by Algorithm 1) and $\mathcal{D}_{steps}$ (computed by Algorithm 2) relatively to the canonical distribution, for a skewed initial distribution and for both metrics.

4.3.2 Skewed balanced initial data distribution

Real world data distributions are usually not random. Some data are more likely to be initially hosted by some particular processor. In this section, we show the possible gain of using the proposed algorithms for skewed initial distributions. We consider a balanced target data partition $\mathcal{P}_{tar}$ where each of its P components includes D elements of data. For $0 \leq \alpha \leq 1$, we note $\mathcal{D}^{\alpha}_{ini}$ the initial data distribution which maps $\lfloor \alpha D \rfloor$ data items of the j^{th} component of $\mathcal{P}_{tar}$ on processor $(j+1) \mod P$, and which randomly maps the other $D - \lfloor \alpha D \rfloor$ data items of this component to all P processors. Note that $\mathcal{D}^{0}_{ini}$ represents a random balanced data distribution as studied in previous section.

We still use $\mathcal{D}_{can}$, the arbitrary target distribution which maps the j^{th} component of $\mathcal{P}_{tar}$ onto processor j, as a comparison basis. During the redistribution phase from $\mathcal{D}^{\alpha}_{ini}$ to $\mathcal{D}_{can}$, each processor sends at least $\lfloor \alpha D \rfloor$ of its elements. With the skewed distribution, we can compute the expected volume of communications of $\mathcal{D}_{can}$: $E(\mathit{RedistVol}(\mathcal{D}^{\alpha}_{ini} \rightarrow \mathcal{D}_{can})) = D(P-1) + \lfloor \alpha D \rfloor$. The number of steps required to schedule the redistribution phase from $\mathcal{D}^{\alpha}_{ini}$ to $\mathcal{D}_{can}$ is equal to D with probability $1 - \left(1 - \left(\frac{P-1}{P}\right)^{D - \lfloor \alpha D \rfloor}\right)^{P}$.

Figure 3 depicts the relative volume of communication and the relative number of redistribution steps for the target distributions $\mathcal{D}_{vol}$ (obtained with Algorithm 1) and $\mathcal{D}_{steps}$ (obtained with Algorithm 2), normalized with the performance of the arbitrary target distribution $\mathcal{D}_{can}$. The simulations have been conducted with $P = 32$ processors, $D = 20$ elements of data on each of them and α varying from 0 to 1. When α is close to 0, $\mathcal{D}^{\alpha}_{ini}$ is close to a random balanced data distribution and we retrieve the results of the previous section. When α is larger than 0.2, for any j, the proportion of data in the j^{th} component of $\mathcal{P}_{tar}$ that are initially hosted by processor $(j+1) \mod P$ is significant. Thus, mapping this component onto processor $(j+1) \mod P$ becomes the best solution to reduce both the volume of communication and the number of communication steps. In this case, Algorithm 1 and Algorithm 2 provide the same target data distribution. Both objectives decrease linearly with α since the proportion of data that are initially mapped onto the correct processor increases linearly with α.

5 Coupling redistribution and stencil computations

In this section, we focus on a simple, yet realistic, application to assess the complexity of redistribution when coupled to a computational kernel. We consider a 1D-stencil iterative algorithm, which updates the elements of an array in parallel, according to the value of their direct neighbors. Stencil computations are widely used to numerically solve partial differential equations [24]. We first detail the application model before establishing the NP-completeness of minimizing the cost of a redistribution followed by the

execution of the kernel.

5.1 Application model

We consider here a three-point stencil with circular arrangement of the data. More precisely, to compute the value $x(i,t)$ of the data at position i at step t, we need its value and those of its left and right neighbors at the previous step, namely $x(i,t-1)$, $x(i-1 \mod N, t-1)$, and $x(i+1 \mod N, t-1)$. If the neighbors are not stored on the same processor, their value has to be received from the processors hosting them. Thus, each iteration of the stencil algorithm consists in two phases, the *communication phase* when the value of each data item is sent to the processors hosting its neighbors, and the *computation phase*, when each data item is updated according to a given kernel using these values (see Algorithm 3). The update kernel depends on the application.

Algorithm 3: One iteration of the unidimensional stencil algorithm

```
Result: N data items numbered from 0 to N − 1 and their distribution D on P processors
for 0 ≤ x ≤ N − 1 in parallel do
    ℓ_x ← (x − 1) mod N;
    r_x ← (x + 1) mod N;
    if D(ℓ_x) ≠ D(x) then
        Processor D(x) receives data item ℓ_x from processor D(ℓ_x);
    if D(r_x) ≠ D(x) then
        Processor D(x) receives data item r_x from processor D(r_x);
for 0 ≤ x ≤ N − 1 in parallel do
    Processor D(x) updates data item x using ℓ_x and r_x;
```

Given a data partition $\mathcal{P}_{tar}$, let N_{ij} be the number of data items sent by the processor hosting the i^{th} component of $\mathcal{P}_{tar}$ to the processor hosting the j^{th} component during one communication phase of the stencil algorithm: N_{ij} is the number of left or right neighbors in the i^{th} component of data items in the j^{th} component. Formally:

$$N_{ij} = |\{0 \leq x \leq N-1 \text{ s.t. } \mathcal{P}_{tar}(x) = i \text{ and } (\mathcal{P}_{tar}(x-1 \mod N) = j \text{ or } \mathcal{P}_{tar}(x+1 \mod N) = j)\}|.$$

The workload ℓ_i of the processor i hosting the i^{th} component of $\mathcal{P}_{tar}$ is $\ell_i = |\{0 \leq x \leq N-1 \text{ s.t. } \mathcal{P}_{tar}(x) = i\}|$.

Given a data partition $\mathcal{P}_{tar}$, the running time of the stencil algorithm depends on the communication model, but not on the actual data distribution, provided that it is compatible with $\mathcal{P}_{tar}$. Let τ_{comm} be the time needed to perform one communication (see Section 3.3), and let τ_{calc} be the time needed to perform one data update for the considered stencil application. The processing time for K iterations of the stencil with the two communication models is the following (using the notations of Section 3.3):

- **Total volume:** For problem VolPart&Redistrib, $T_{comp}(\mathcal{P}_{tar}) = K \times T^{iter}_{vol}(\mathcal{P}_{tar})$, where

$$T^{iter}_{vol}(\mathcal{P}_{tar}) = \tau_{comm} \times \left(\sum_{0 \leq i \leq P-1} \sum_{j \neq i} N_{ij} \right) + \tau_{calc} \times \max_{0 \leq i \leq P-1} \ell_i .$$

  The first term corresponds to the serialization of all communications, and the second one to the parallel processing of the updates.

- **Number of parallel steps:** For problem StepPart&Redistrib, $T_{comp}(\mathcal{P}_{tar}) = K \times T^{iter}_{steps}(\mathcal{P}_{tar})$, where

$$T^{iter}_{steps}(\mathcal{P}_{tar}) = \tau_{comm} \times \max_{0 \leq i \leq P-1} \left(\sum_{j \neq i} N_{ij}, \sum_{j \neq i} N_{ji} \right) + \tau_{calc} \times \max_{0 \leq i \leq P-1} \ell_i .$$

  Here the first term corresponds to the time needed to perform the required number of communication steps, and the second term is unchanged.

5.2 Complexity

Assume without loss of generality that N is a multiple of P. There is a well-known optimal data partition for the 1D-stencil kernel, namely the full block partition (data item i is assigned to component $\lfloor iP/N \rfloor$). This *canonical* partition $\mathcal{P}_{can}$ minimizes the duration of the communication phase (only two items are sent/received per component of the partition) and the computation phase is perfectly balanced.

Starting from an initial data distribution $\mathcal{D}_{ini}$, we can use either Algorithm 1 or 2 to find a target distribution $\mathcal{D}_{tar}$ which is compatible with the full-block partition $\mathcal{P}_{can}$ and whose redistribution cost is minimal. However, redistributing from $\mathcal{D}_{ini}$ to $\mathcal{D}_{tar}$ may induce a large overhead on the total execution time, which is fully justified only when the number of iterations K is large enough. It may be useful to avoid a costly redistribution for small values of K, and to find a target redistribution which is a trade-off between minimizing redistribution time and processing time. Actually, finding such a trade-off distribution is an NP-complete problem for both communication models. We define the two decision problems associated to VolPart&Redistrib and StepPart&Redistrib:

Definition 8 (DecisionVolPart&Redistrib)**.** Given a number of processors P, elementary communication and computation times τ_{comm} and τ_{calc}, a number of steps K, an initial data distribution $\mathcal{D}_{ini}$ and a bound T_{MAX}, are there a partition $\mathcal{P}_{tar}$, and a distribution $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$, such that: $T_{total}(\mathcal{D}_{ini}, \mathcal{D}_{tar}) = \mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm} + T_{comp}(\mathcal{P}_{tar}) \leq T_{MAX}$?

Definition 9 (DecisionStepPart&Redistrib)**.** Given a number of processors P, elementary communication and computation times τ_{comm} and τ_{calc}, a number of steps K, an initial data distribution $\mathcal{D}_{ini}$ and a bound T_{MAX}, are there a partition $\mathcal{P}_{tar}$, and a distribution $\mathcal{D}_{tar}$ compatible with $\mathcal{P}_{tar}$, such that: $T_{total}(\mathcal{D}_{ini}, \mathcal{D}_{tar}) = \mathit{RedistSteps}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm} + T_{comp}(\mathcal{P}_{tar}) \leq T_{MAX}$?

Theorem 3. *The* DecisionVolPart&Redistrib *problem with the 1D-stencil kernel is strongly NP-complete.*

Proof. We first prove that DecisionVolPart&Redistrib belongs to NP. Given $\mathcal{D}_{tar}$, it is possible to compute in polynomial time the redistribution time $\mathit{RedistVol}(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) \times \tau_{comm}$ and the cost of the K iterations of the stencil algorithm $T_{comp}(\mathcal{P}_{tar})$, and thus to check whether T_{total} is smaller than T_{MAX} or not. Thus, DecisionVolPart&Redistrib is in NP.

To establish the completeness, we use a reduction from the 3-Partition problem, which is known to be NP-complete in the strong sense [25]. We consider the following instance Inst_0 of the 3-Partition problem: let a_i^0, $1 \leq i \leq 3m$, be $3m$ integers and B^0 an integer such that $\sum a_i^0 = mB^0$. We enforce the additional (usual) constraint that $\forall i, B^0/4 < a_i^0 < B^0/2$ [25]. To solve Inst_0, we need to solve the following question: is there a partition of the a_i^0's in m subsets $S_1, ..., S_m$ such that, $\forall S_k, \sum_{i \in S_k} a_i^0 = B^0$. Note that if there is a solution, then each subset will contain exactly 3 elements, due to the additional constraint.

We then transform (in polynomial time) Inst_0 into another instance Inst_1 of the 3-Partition problem as follows: let $a_i = 385m \times a_i^0$ for $1 \leq i \leq 3m$ and $B = 385m \times B^0$. Obviously, Inst_1 has a solution if and only if Inst_0 has a solution. This new instance of the 3-Partition problem has the following properties:

- $\forall \mathbf{i}, \mathbf{B/4 < a_i < B/2}$, since $\forall i, B^0/4 < a_i^0 < B^0/2$
- $\mathbf{B^2 > 5m \times B + 96m}$ since $\frac{1}{m}(B^2 - 5m \times B) = 385mB^0(385B^0 - 5) > 96$ because $m \geq 1$ and $B^0 \geq 1$.
- $\mathbf{B > 384m}$ since $B^0 \geq 1$.

Given Inst_1, we build the following instance Inst_2 of the DecisionVolPart&Redistrib problem, illustrated in Figure 4. In Inst_2, we set the number of processors to $P = 12m$, the number of 1D-stencil steps to $K = 1$, elementary communication and computing times $\tau_{comm} = 1$ and $\tau_{calc} = B^2$. We also set the time bound to $T_{\text{MAX}} = 96m + 5mB + 8B^3$. Finally, Figure 4 represents the initial data distribution $\mathcal{D}_{ini}$ of $96mB$ elements on the $12m$ different processors. To clarify the proof, we split the $12m$ processors into 4 different groups. There are $3m$ processors in group 1, m processors in group 2, $4m$ processors in group 3 and $4m$ processors in group 4. Processors in group k are denoted by $P_i^{(k)}$. Figure 4 depicts the initial data distribution $\mathcal{D}_{ini}$. For example, the $2B$ first consecutive elements are stored on $P_1^{(1)}$, the first processor in group 1. The next $2B$ elements are stored on $P_1^{(4)}$, the first processor in group 4. Note that in the fifth set of $3m$ block values (a_1, $2B$, a_2, $2B$, ..., a_{3m}, $2B$), the $3m$ blocks of size $2B$ are distributed on the m group-4 processors $P_{3m+1}, \ldots, P_{4m}$ in a round robin way (the first block goes to P_{3m+1}, the second one to P_{3m+2}, ..., the m-th block goes to P_{4m}, the $m+1$-th goes ot P_{3m+1}, etc.

The construction of $Inst_2$ is polynomial in the size of $Inst_1$, and thus, in the size of $Inst_0$. We show that $Inst_2$ has a solution if and only if $Inst_1$ has a solution.

We first assume that $Inst_2$ has a solution and we let $\mathcal{D}_{tar}$ denote be the final distribution of data. A *connected component* of processor p is defined as a set of consecutive items hosted on processor p. A *maximal connected component* of processor p is a connected component of processor p which is not strictly included in another connected component of processor p. For instance, in $\mathcal{D}_{ini}$ depicted in Figure 4, each group-1 processor has 5 maximal connected components in $\mathcal{D}_{ini}$. Let C_p be the number of maximal connected components on processor p for the distribution $\mathcal{D}_{tar}$. At each stencil step, each processor has to send only the two items at each border of each of its maximal connected components. Thus, with $l_p = |\{x \text{ s.t. } \mathcal{D}_{tar}(x) = p\}|$ being the workload of processor p as introduced in Section 5.1, $T^{iter}_{vol}(\mathcal{P}_{tar}) = 2 \times \sum_p C_p + B^2 \times \max_p l_p$, and $T_{total}(\mathcal{D}_{ini}, \mathcal{D}_{tar}) = RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) + 2 \times \sum_p C_p + B^2 \times \max_p l_p$. Since, $\mathcal{D}_{tar}$ is a solution to $Inst_2$, we know that:

$$RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) + 2 \times \sum_p C_p + B^2 \times \max_p l_p \leq 96m + 5mB + 8B^3. \quad (1)$$

Let us first show that $\forall \mathbf{p},\ \mathbf{l_p = 8B}$:

- $\max_p l_p \leq 8B$ because otherwise, we would have:

$$T^{stencil}_{vol}(\mathcal{D}_{ini}, \mathcal{D}_{tar}) \geq B^2 \times \max_p l_p \geq B^2 \times (8B + 1) > T_{\text{MAX}},$$

  since $B^2 > 5m \times B + 96m$.

- There are a total of $96mB$ elements of data and $12m$ processors, thus $\forall p,\ l_p = 8B$.

Thus $\max_p l_p = 8B$ and Equation 1 becomes:

$$RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) + 2 \times \sum_p C_p \leq 96m + 5mB. \quad (2)$$

For each processor $P_i^{(k)}$, let $S_i^{(k)}$ (respectively $R_i^{(k)}$) be the number of elements sent (respectively received) by processor $P_i^{(k)}$ during the redistribution phase. We naturally have

$$\sum_{k,i} S_i^{(k)} = \sum_{k,i} R_i^{(k)} = RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar}).$$

Let us show that $\sum_p C_p \geq 48m$. Initially, in $\mathcal{D}_{ini}$, there are $52m$ maximal connected components among all the processors. There are only two different ways to decrease the global number of maximal connected components: merging two existing connected components by receiving all the data between them, or sending one entire maximal connected component to one of the processors that host a maximal connected component next to it. We first consider the first option. In $\mathcal{D}_{ini}$, two maximal connected components hosted by the same processor are separated by more than $6mB$ elements. Thus, to merge two existing maximal connected components in $\mathcal{D}_{ini}$, a processor would have to receive more than $6mB$ elements during the redistribution phase, which is not possible according to Equation 2, since $B > 384m$. We now consider the second option (sending one entire maximal connected component).

- Let assume that a processor $P_i^{(1)}$ sends one of its entire maximal connected components to one of its neighbors. The only neighbors of $P_i^{(1)}$ are processors of group-4. This means that a processor $P_j^{(4)}$ will receive at least $B/4$ elements from $P_i^{(1)}$ during the redistribution phase, since $\forall i, B/4 < a_i$. However, at the end of the redistribution phase, $P_j^{(4)}$ can only host $8B$ elements, thus it will have to send at least $\frac{5}{4}B$ elements during the redistribution phase:

$$RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = \sum_{(k,p) s.t. (k,p) \neq (4,j)} S_p^{(k)} + S_j^{(4)} \geq 5mB + B/4,$$

  which is not possible according to Equation 2 and since $B > 384m$.

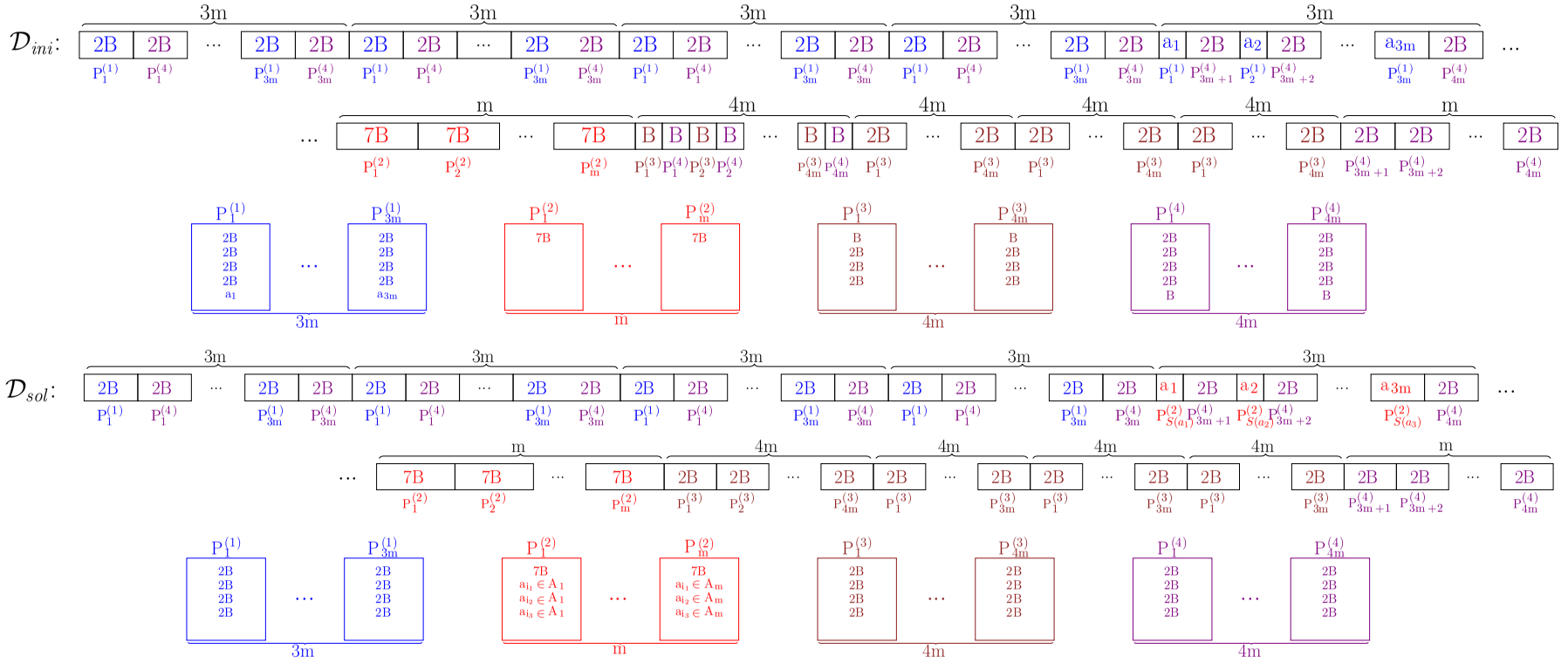


Figure 4: $\mathcal{D}_{ini}$ and $\mathcal{D}_{sol}$ in the proof of Theorems 3 and 4.

- Let assume that a processor $P_i^{(2)}$ sends one of its entire maximal connected components to one of its neighbors. This means that processor $P_i^{(2)}$ will send at least $7B$ elements during the redistribution phase and thus, we will have $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = \sum_{(k,p)s.t.(k,p)\neq(2,i)} S_p^{(k)} + S_i^{(2)} \geq 5mB + 7B$, which again, is not possible according to Equation 2 and since $B > 384m$.

- Let assume that a processor $P_i^{(3)}$ sends one of its entire maximal connected components to one of its neighbors. This means that $P_i^{(3)}$ will send at least B elements during the redistributing phase and thus, we will have $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = \sum_{(k,p)s.t.(k,p)\neq(3,i)} S_p^{(k)} + S_i^{(3)} \geq 5mB + B$, which, again, is not possible.

Thus, the only remaining option to decrease the number of maximal connected component is that some processor $P_i^{(4)}$ sends at least one entire connected component to one of its neighbors. Assume that it sends at least two of its entire maximal connected components to one of its neighbors. This means that $P_i^{(4)}$ will send at least $3B$ elements during the redistributing phase and thus, we will have $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = \sum_{(k,p)s.t.(k,p)\neq(3,i)} S_p^{(k)} + S_i^{(4)} \geq 5mB + 2B$, which, again, is not possible according to Equation 2 and since $B > 384m$.

Thus each processor $P_i^{(4)}$ can send only one of its entire maximal connected components to one of its neighbors. There are $4m$ processors in group-4, so we can reduce the number of maximal connected components by only $4m$. Since there are $52m$ maximal connected components in $\mathcal{D}_{ini}$, we have in $\mathcal{D}_{tar}$:

$$\sum_p C_p \geq 48m. \tag{3}$$

Then, we show that $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = 5mB$.

- Equation 2 and Equation 3 lead to: $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = \sum_{k,i} R_i^{(k)} \leq 5mB$

- Initially, in $\mathcal{D}_{ini}$, each processor in group-2 or group-3 hosts $7B$ elements of data and since $\forall p$, $l_p = 8B$ in $\mathcal{D}_{tar}$, the group-2 and group-3 processors each have to receive at least B elements of data during the redistribution phase ($\forall i, R_i^{(2)} \geq B$ and $R_i^{(3)} \geq B$). Thus at least $5mB$ elements of data have to be communicated during the redistribution phase: $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = \sum_{k,i} R_i^{(k)} \geq 5mB$.

Thus $RedistVol(\mathcal{D}_{ini} \rightarrow \mathcal{D}_{tar}) = 5mB$ and Equation 2 becomes:

$$\sum_p C_p = 48m. \tag{4}$$

We now bound the number of elements sent and received by processors in group 2, 3 and 4.

- Each group-2 processor $P_i^{(2)}$ and each group-3 processor $P_i^{(3)}$ hosts $7B$ elements in the initial distribution $\mathcal{D}_{ini}$, and $8B$ elements in the final distribution $\mathcal{D}_{tar}$. Thus, they each have to receive at least B elements of data. There are $5m$ of them, so they can receive only B elements of data each, and no other processor can receive any data. More formally, we have $\forall i$: $R_i^{(1)} = 0$, $R_i^{(2)} = B$, $R_i^{(3)} = B$ and $R_i^{(4)} = 0$.

- Each group-1 processor $P_i^{(1)}$ hosts $8B + a_i$ elements in $\mathcal{D}_{ini}$, and $8B$ elements in $\mathcal{D}_{tar}$. Each group-4 processor $P_i^{(4)}$ hosts $9B$ elements in $\mathcal{D}_{ini}$, and $8B$ elements in $\mathcal{D}_{tar}$. Again, this means that each group-1 processor $P_i^{(1)}$ can send only a_i elements of data, each group-4 processor $P_i^{(4)}$ can send only B elements of data and no other processors can send any data. That is, $\forall i$: $S_i^{(1)} = a_i$, $S_i^{(2)} = 0$, $S_i^{(3)} = 0$ and $S_i^{(4)} = B$.

Since $\sum_p C_p = 48m$, each group-4 processor has to send one and only one of its entire maximal connected components to one of its neighbor (as we have seen earlier, there is no other way to decrease the global number of maximal connected components). Since $\forall i$: $S_i^{(4)} = B$, group-4 processors can only

send their maximal connected component of size B. The only neighbors of these maximal connected components are some group-3 processors. Thus, each group-4 processor will send its entire maximal connected component of size B to a group-3 processor, and group-3 processors can not receive anything else from any other processor.

Gathering all the results shown above, we can state that group-1 processors can only send their a_i elements to group-2 processors during the redistribution phase. If a processor $P_i^{(1)}$ splits its a_i consecutive elements and send them to two different group-2 processors, this would create an extra maximal connected component on the group-2 processors. Thus, **each group-1 processor has to send its $\mathbf{a_i}$ elements to the same group-2 processor**.

Let A_k be the set of the sizes of the maximal connected components received by $P_k^{(2)}$ during the redistribution phase. The A_k sets represent a partition of the a_i's and $\forall k, \sum_{a_i \in A_k} a_i = R_k^{(2)} = B$. Hence the A_k sets are a solution of $Inst_1$.

Suppose now that $Inst_1$ has a solution. Let A_k be the 3-Partition of the integers a_i and consider the distribution $\mathcal{D}_{sol}$ described in Figure 4. To perform the redistribution from $\mathcal{D}_{ini}$ to $\mathcal{D}_{sol}$, each group-2 and group-3 processors has to send or receive B elements of data, which means that $RedistVol(\mathcal{D}_{ini} \to \mathcal{D}_{sol}) = 5mB$. In addition, in $\mathcal{D}_{sol}$, there are $48m$ maximal connected components. Thus, $T_{total}(\mathcal{D}_{ini}, \mathcal{D}_{sol}) = 96m + 5mB + 8B^3 \leq T_{MAX}$, which means that $Inst_2$ has a solution. This concludes the proof. □

Theorem 4. StepPart&Redistrib *problem with the 1D-stencil kernel is strongly NP-complete.*

Proof. The proof of Theorem 4 is similar to that of Theorem 3. We consider the same instances $Inst_0$ and $Inst_1$ of the 3-Partition problem [25]. We build the instance $Inst_2$ depicted in Figure 4 for the DecisionStepPart&Redistrib problem, as in the previous proof, except that we now set $T_{\text{MAX}} = 8 + B + 8B^3$. We want to show that $Inst_2$ has a solution if and only if $Inst_1$ has a solution.

Assume first that $Inst_2$ has a solution and use the same notations as above. We have the inequality:

$$\begin{aligned} T_{total}(\mathcal{D}_{ini}, \mathcal{D}_{tar}) &= RedistSteps(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) + 2 \times \max_p C_p + B^2 \times \max_p l_p \\ &\leq 8 + B + 8B^3. \end{aligned} \tag{5}$$

As in the previous proof, we can easily show that $\forall p,\ l_p = 8B$. Thus, Equation 5 becomes:

$$RedistSteps(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) + 2 \times \max_p C_p \leq 8 + B. \tag{6}$$

Then we show that $RedistSteps(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = \max_{(p,k)} max(S_p^{(k)}, R_p^{(k)}) = B$:

- Initially, in $\mathcal{D}_{ini}$, the processor $P_1^{(4)}$ hosts $9B$ elements of data; since $\max_p l_p = 8B$ in $\mathcal{D}_{tar}$, the processor $P_1^{(4)}$ has to send at least B elements of data during the redistribution phase. Thus, $S_1^{(4)} \geq B$ and $RedistSteps(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) \geq B$.
- If one processor sends $B + 1$ elements of data during the redistribution phase, we would have $RedistSteps(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) \geq B + 1$ and $2 \times \max_p C_p \leq 7$, so $\max_p C_p \leq 3$. Initially, in $\mathcal{D}_{ini}$, processor $P_1^{(1)}$ hosts 5 maximal connected components and could have at most 3 maximal connected components in $\mathcal{D}_{tar}$. There are only two different ways to decrease the number of maximal connected components in a processor: sending one entire maximal connected component to another processor or merging two existing connected components by receiving all the data between them. Both options are impossible in this case, because processor $P_1^{(1)}$ would have to send or receive more than $2B$ elements during the redistribution phase, which is impossible according to Equation 5.

Thus $RedistSteps(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = B$ and Equation 6 becomes:

$$\max_p C_p \leq 4. \tag{7}$$

We now bound the number of elements sent and received by processors in group 2, 3 and 4. We naturally have

$$\forall P_i^{(k)},\ max(S_i^{(k)}, R_i^{(k)}) \leq RedistSteps(\mathcal{D}_{ini} \to \mathcal{D}_{tar}) = B.$$

- Each group-2 processor $P_i^{(2)}$ hosts $7B$ elements in the initial distribution $\mathcal{D}_{ini}$, and $8B$ elements in the final distribution $\mathcal{D}_{tar}$. This means that $R_i^{(2)} - S_i^{(2)} = B$. Since $max(S_i^{(2)}, R_i^{(2)}) \leq B$, we have: $\mathbf{R_i^{(2)}} = \mathbf{B}$ and $\mathbf{S_i^{(2)}} = \mathbf{0}$.

- Each group-3 processor $P_i^{(3)}$ hosts $7B$ elements in $\mathcal{D}_{ini}$, and $8B$ elements in $\mathcal{D}_{tar}$. Again, this means that $R_i^{(3)} - S_i^{(3)} = B$ and since $max(S_i^{(3)}, R_i^{(3)}) \leq B$, we necessarily have: $\mathbf{R_i^{(3)}} = \mathbf{B}$ and $\mathbf{S_i^{(3)}} = \mathbf{0}$.

- Each group-4 processor $P_i^{(4)}$ hosts $9B$ elements in $\mathcal{D}_{ini}$, and $8B$ elements in $\mathcal{D}_{tar}$. Again, $S_i^{(4)} - R_i^{(4)} = B$ and we have $\mathbf{R_i^{(4)}} = \mathbf{0}$ and $\mathbf{S_i^{(4)}} = \mathbf{B}$.

From these results, using the same reasoning as in the previous proof, we can show that:

- group-1 and group-2 processors do not send or receive any data to or from a group-3 or group-4 processor.

- Each group-1 processor does not keep any data received during the redistribution phase.

- Each group-1 processor has to send its $\mathbf{a_i}$ elements to the same group-2 processor.

Let A_k be the set of the sizes of the maximal connected components received by $P_k^{(2)}$ during the redistribution phase: we have shown that the A_k sets are a solution of $Inst_1$.

Suppose now that $Inst_1$ has a solution. As in the previous proof, we can show that the distribution $\mathcal{D}_{sol}$ described in Figure 4 is a solution for $Inst_2$, which concludes the proof.

□

6 Experiments

The algorithms designed in Section 4 find the optimal target distribution according to different models for the redistribution time. These algorithms may be sub-optimal for minimizing the total processing time when it takes the processing of an arbitrary application into account. Section 5 proved that there is no polynomial-time optimal algorithm to minimize this total processing time (unless P=NP) even for a simple application like the 1D-Stencil algorithm, which motivates the use of low-complexity sub-optimal heuristics. In this section, we show that the redistribution algorithms introduced in Section 4 are good enough to provide performance improvements in real-life applications. The experiments are conducted on a multicore cluster for the 1D-Stencil kernel and, then, for a more compute-intensive dense linear algebra routine, namely the QR factorization.

6.1 Setup

We have implemented the 1D-stencil kernel of Section 5 on top of the PaRSEC runtime [26, 27]. In addition, we have also implemented a QR factorization algorithm on top of PaRSEC, in order to experiment with a widely used computation-intensive numerical linear algebra routine.

The PaRSEC runtime deals with computational threads and MPI communications. It allows the user to define the initial distribution of the data onto the platform, as well as the target distribution for the computations. Data items are first moved from their initial data distribution to the target data distribution. Then computations take place, and finally data items are moved back to their initial position. It is important to stress that the PaRSEC runtime will overlap the initial communications due to the redistribution with the processing of the computational kernel (either 1D-stencil or QR), so that the total execution time does not strictly obey the simplified model of the previous sections. However, choosing a good data partition (leading to an efficient implementation of the computational kernel), and an efficient compatible data distribution (leading to fewer communications during the redistribution) is still important to achieve high performance.

Experiments have been conducted on *Dancer*, a small cluster hosted at the Innovative Computing Laboratory (ICL) in Knoxville, TN. This cluster has 16 multi-core nodes, each equipped with 8 cores,

and an InfiniBand 10G interconnection network. Each node features two Intel Westmere-EP E5606 CPUs at 2.13GHz. The system is running the Linux 64bit operating system, version 3.7.2-x86_64. The software was compiled with the Intel Compiler Suite 2013.3.163. BLAS kernels were provided by the MKL library, and OpenMPI 1.4.3 was used for MPI communications by the PaRSEC runtime version 1.1. Each computational thread is bound to a single core using the HWLOC 1.7.1 library. We use all 16 nodes, whose aggregated theoretical peak performance is $1,091$ GFLOP/sec.

6.2 Stencil

The stencil algorithm described in Algorithm 3 can use diverse patterns to update the data, depending upon the target application. In our experiments, data items operated upon are blocks of 1.6×10^6 double-precision floats. Experimentally, we observe that the average communication time of such blocks between two nodes of *Dancer* is 100 milliseconds. The data items are initially distributed on the 16 processors according to a random balanced distribution as described in Section 4.3. We used a set of 30 randomly generated initial data distributions.

Figure 5 depicts the performance of the 1D-stencil algorithm when the update kernel takes on average 100 milliseconds to compute the new value of one data item, so that the communication-to-computation ratio is $\tau_{comm}/\tau_{calc} = 1$. Each sub-figure represents a different number of stencil iterations ($K = 0$ to 9). In each sub-figure, we have executed K stencil iterations with 4 different strategies. In the *owner computes* strategy, the data items are not moved and the stencil algorithm is applied on the initial distribution. In the other strategies, we redistribute the data items towards three target distributions, each compatible with the canonical data partition $\mathcal{P}_{can}$ described in Section 5.2: (i) the distribution $\mathcal{D}_{can} = \mathcal{P}_{can}$ with the original (arbitrary) labeling of the processors; (ii) the distribution that minimizes the volume of communications $\mathcal{D}_{vol}$; and (iii) the distribution that minimizes the number of redistribution steps $\mathcal{D}_{steps}$. We compute the processing time of the redistribution followed by the K stencil iterations. The time needed to compute the target distributions depends on the number of processors and data items but does not depend on the size of the data items. Usually the size of the data items is large enough for the computation time of the algorithm presented in Section 4 to be negligible, therefore it is not included in the figures. Each cross shows the performance for one of the 30 initial data distributions, and the plain lines shows the average performance on the 30 initial data distributions. The first observation is that the standard deviation of the processing time for all the initial data distribution is very small. Moreover, in all sub-figures, we observe that the performances for target distributions $\mathcal{D}_{vol}$ and $\mathcal{D}_{steps}$ are indistinguishable. This is in accordance with the results in Section 4.3 showing that, on random balanced initial distributions, Algorithm 1 and Algorithm 2 provide similar performances for both metrics. We observe that the processing time of the three redistribution strategies slightly increases with the number of stencil iterations, i.e., one stencil iteration is very fast (roughly 400 milliseconds for 16 data items per processor) when processed on the optimal data partition described in Section 5.2. However, the *owner computes* strategy is less efficient as soon as we have to process more than one stencil iteration. In the top-left sub-figure, $K = 0$ so that no iteration is executed. Data items are moved from their initial processor to their target processor and then moved back onto their initial position. It thus depicts the performance of two consecutive redistributions. The *owner computes* strategy has a processing time close to zero which corresponds to the overhead of the PaRSEC runtime. Both redistribution strategies computed by Algorithm 1 and Algorithm 2 provide a 20% improvement over $\mathcal{D}_{can}$. This improvement decreases when the number of iterations increases. Indeed, the only difference between the performances of $\mathcal{D}_{can}$, $\mathcal{D}_{vol}$ and $\mathcal{D}_{steps}$ comes from the redistribution phase: the heavier the computation, the less significant the redistribution phase.

Figure 6 depicts the performance of the 1D-stencil algorithm when the update kernel is less expensive, so that $\tau_{comm}/\tau_{calc} = 10$. Hence, in this experiment, the cost of communicating a data element is greater than the computation time and we have to take special care to the redistribution. With a faster computing kernel, the overall computation time is inferior to the one in Figure 5 but the *owner computes* strategy is still less efficient than the redistributing strategies as soon as we have to do more than one iteration. The difference between the performances of $\mathcal{D}_{can}$, $\mathcal{D}_{vol}$ and $\mathcal{D}_{steps}$ is smaller in percentage than in Figure 5.

Altogether, the experiments show that (i) redistributing towards a better data distribution is more suitable than performing the algorithm in place with the random initial distribution, as soon as the computational cost in non-negligible; and (ii) redistributing towards a data distribution that minimizes the cost of the redistribution phase rather than towards an arbitrary one does improve the performance,

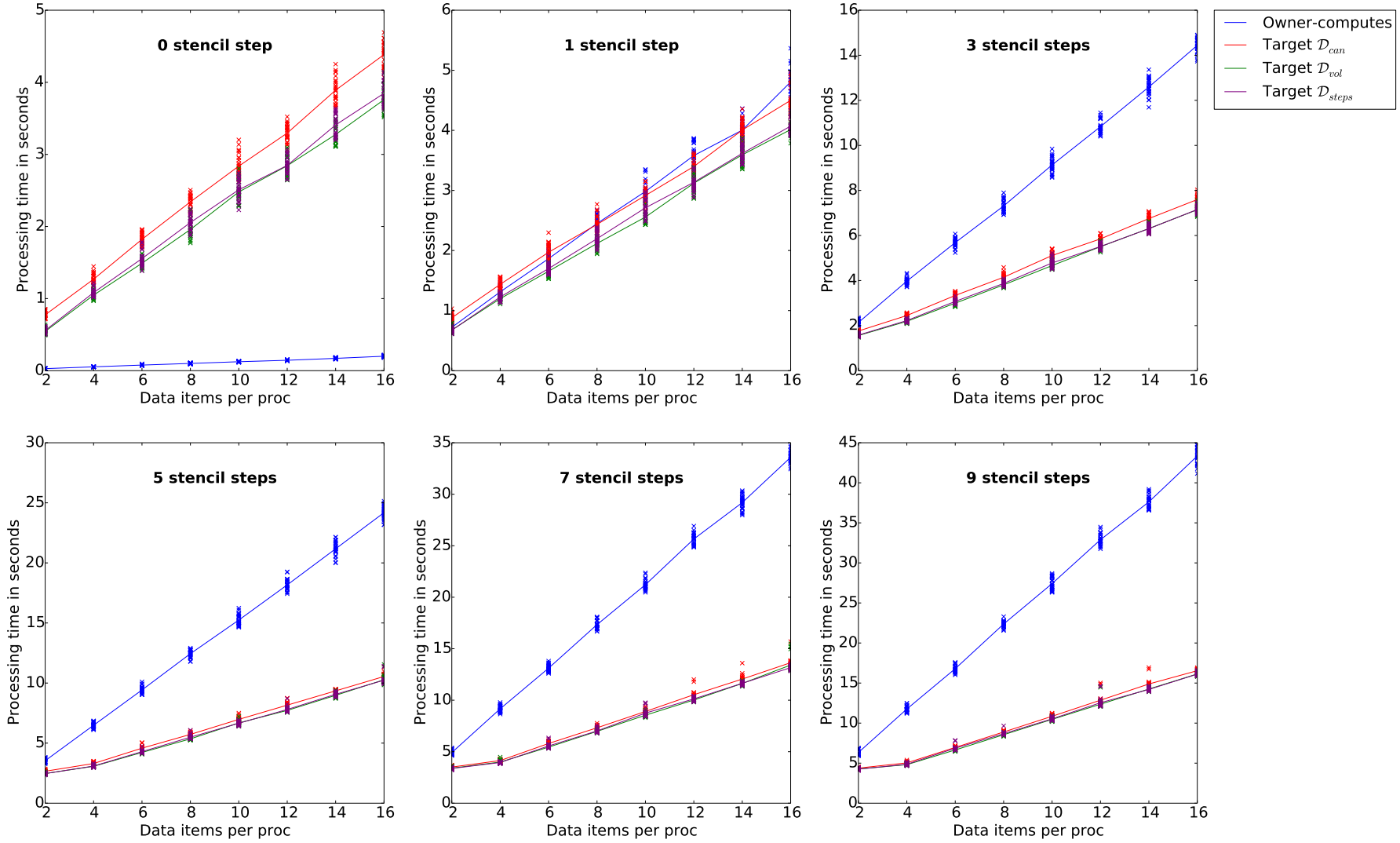


Figure 5: Performance of the stencil algorithm for $\tau_{comm}/\tau_{calc} = 1$.

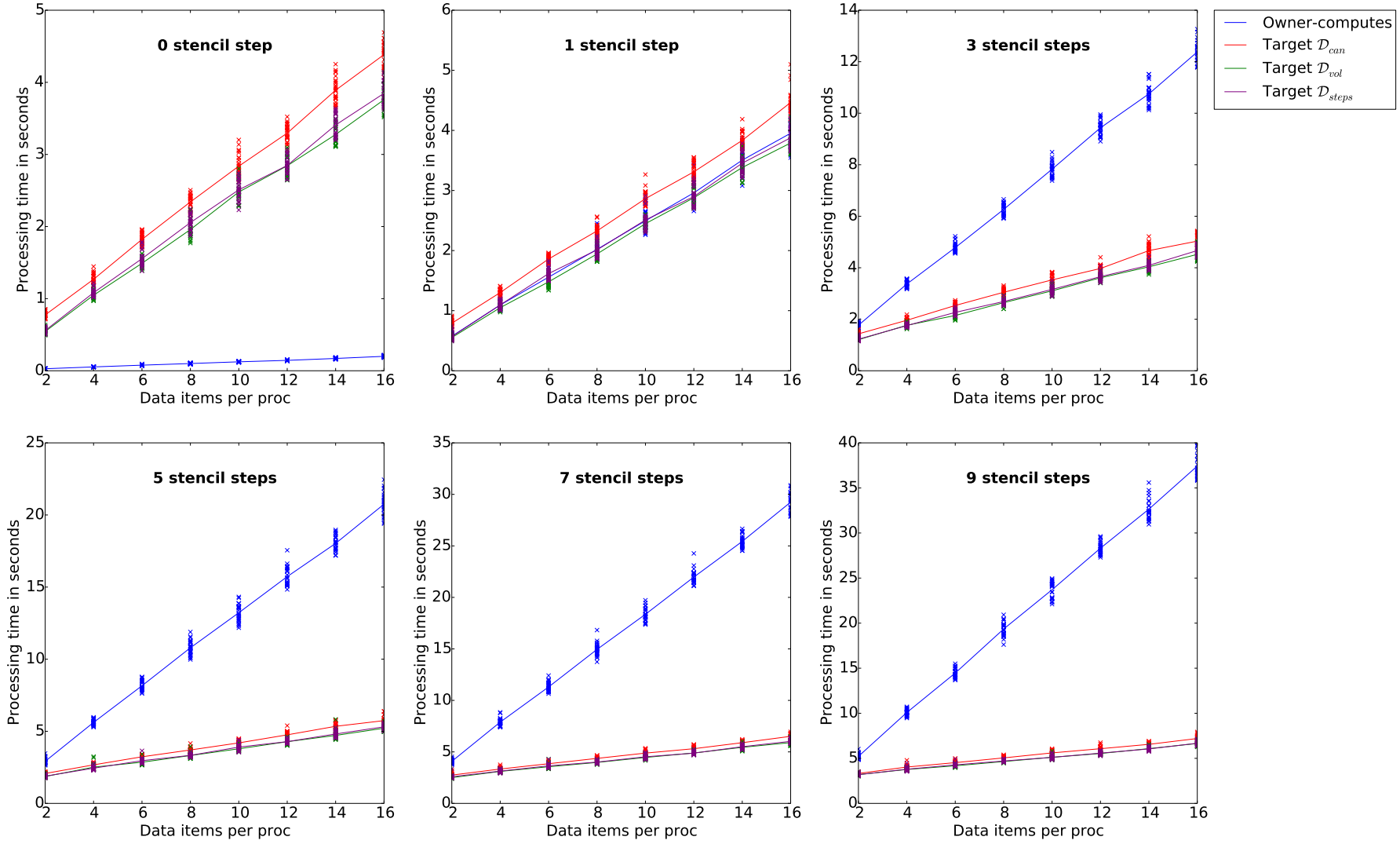


Figure 6: Performance of the stencil algorithm for $\tau_{comm}/\tau_{calc} = 10$.

especially when the time to communicate a data item is significant.

6.3 QR factorization

In this section, we deal with a more compute-intensive kernel, namely the QR factorization, which consists of decomposing a square matrix A into the product of two matrices $Q \times R$ such that Q is an orthogonal matrix and R is an upper triangular matrix. QR factorization is a widely used linear algebra algorithm for solving linear systems and linear least squares problems.

6.3.1 Framework

To optimize performance, the matrix is usually stored in tiled form: A has n tiles per row or column and each tile is a block of $n_b \times n_b$ floating point numbers. The matrix is then factored with a tiled QR factorization algorithm using orthogonal Householder matrices, as in [28, 29]. The $N = n^2$ matrix tiles are the data items of the application. Initially, the tiles are arbitrarily distributed, and this initial distribution may not be suitable for the QR factorization. We aim to redistribute the N data items towards a better data partition. However, as opposed to the 1D-stencil algorithm, the QR factorization algorithm has a complex workflow, and it is impossible to predict its processing time accurately: given a data partition $\mathcal{P}$, we cannot easily compute $T_{comp}(\mathcal{P})$.

However, even though there is no explicit model for the cost of a QR factorization performed on a specific data partition, some distributions are known to be well-suited. A widely-used data partition consists of mapping the tiles onto the processors following a 2D block cyclic partition [30]. The P processors (numbered from 0 to $P-1$) are arranged in a $p \times q$ grid where $p \times q = P$. Matrix tile $A_{i,j}$ is then mapped onto processor $(i \mod p) \times p + (j \mod q)$. In the following, this data partition will be referred to as $\mathcal{P}_{tar}$, and the objective is to redistribute the N data items towards a distribution compatible with $\mathcal{P}_{tar}$.

Similarly to Section 6.2, we compare 4 redistribution strategies. In the *owner computes* strategy, data items are not moved and the QR factorization is performed in place. In the other strategies, we redistribute data items towards three target distributions compatible with $\mathcal{P}_{tar}$: (i) the distribution $\mathcal{D}_{can} = \mathcal{P}_{tar}$ with the original (arbitrary) labeling of the processors; (ii) the distribution that minimizes the volume of communications $\mathcal{D}_{vol}$; and (iii) the distribution that minimizes the number of redistribution steps $\mathcal{D}_{steps}$.

6.3.2 Setup

A highly optimized version of the QR factorization implemented on top of the PaRSEC runtime is available in the DPLASMA library [31]. We have modified this implementation to deal with different data distributions. We use a wide range of matrix sizes, with tiles of size $n_b = 200 \times 200$ double-precision floating point numbers. Our objective is to highlight the impact of the target data distribution on the performance of the QR algorithm, but a tile size of 200×200 is reasonable as it ensures near peak performance on the execution platform.

As already mentioned, real-life distributions are not random. We conduct experiments on 2 different sets of initial distributions for the matrix tiles, one artificially generated and one modeling an Earth Science application [32]:

- *SkewedSet*: Matrix tiles are first distributed following an arbitrary 2D block cyclic distribution (used as reference) and, then, half of the tiles are randomly moved onto another processor. The processor with index $i \in [\![0, P-1]\!]$ has a probability $\frac{2i}{P(P-1)}$ to receive each tile. Thus, the workload among processors is likely to be imbalanced. The redistribution strategies toward $\mathcal{D}_{vol}$ and $\mathcal{D}_{steps}$ should find the 2D block cyclic distribution used as reference and move only half of the tiles, while the redistribution towards the arbitrary distribution $\mathcal{D}_{can}$ can potentially move all of them.

- *ChunkSet*: This distribution set comes from an Earth Science application [32]. Astronomy telescopes collect data over days of observations and process them into a 2D or 3D coordinate system, which is usually best modeled as a matrix. Then, linear algebra routines such as QR factorization must be applied to the resulting matrix. The collected data are stored on a set of processors in a

round-robin manner, ensuring spacial locality of data that are sampled at close time-steps. If a certain region of Earth is observed twice, the latest data overwrites the previous one. We generated a set of initial distributions fitting the telescope behavior. Figure 7 depicts the data distribution of a matrix in *ChunkSet* where matrix tiles of the same color are initially stored on the same processor.

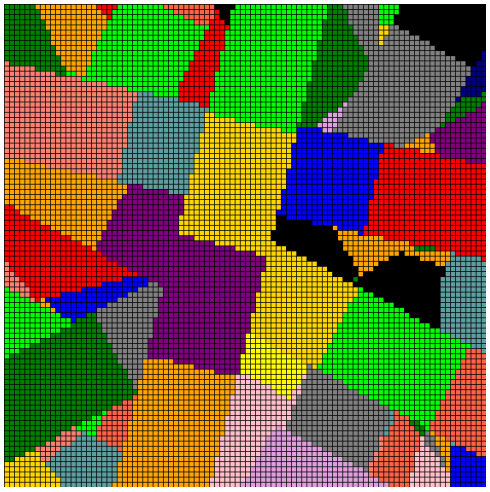


Figure 7: The initial distribution of a tiled matrix in *ChunkSet*.

6.3.3 Results

Table 1a presents the results of the experiments for initial distributions in *SkewedSet*. Each line corresponds to the average results on 50 matrices with $n \times n$ tiles. Columns 1 to 4 give the volume of tiles communicated during the redistribution phase for the four strategies. As expected, redistributing towards the arbitrary distribution $\mathcal{D}_{can}$ requires moving almost every tile while redistributing towards $\mathcal{D}_{vol}$ or $\mathcal{D}_{steps}$ involves almost half as many communications. Columns 5 to 8 present the number of redistribution steps required to schedule the redistribution for the four strategies. We observe that $\mathcal{D}_{vol}$ or $\mathcal{D}_{steps}$ are identical, since Algorithm 1 and Algorithm 2 manage to find the 2D block-cyclic distribution used as reference when building *SkewedSet*. Columns 9 and 10 present the total volume of tiles communicated during the QR factorization. It appears that redistributing towards a 2D block-cyclic partition divides by more than 3 the amount of communications involved in the QR factorization. The gain obtained by redistributing the data according to a suitable partition is significant, and can be seen in the total completion times shown in columns 11 to 14. Columns 15 to 17 present the percentage of improvement provided by the redistribution strategies over the *owner computes* strategy.

Table 1b presents the results of the experiments for initial distributions in *ChunkSet*. Each line corresponds to the average results on 50 matrices, as before. The three redistribution strategies perform similarly, with around 90% of the tiles moved during the redistribution phase. Contrary to the previous case, it appears that redistributing towards a 2D block-cyclic partitioning does not lead to a reduction of the volume of communication involved during the QR factorization. Indeed, the *owner computes* strategy requires fewer communications than the other strategies for larger matrices in *ChunkSet*, due to the chunk distribution of the tiles. However, it does not lead to better performance results. Indeed, the three redistribution strategies require more communications to ensure a better load balancing, which leads to a 10-15% improvement on the total completion times compared to the owner-compute strategy on large matrices.

In summary, we conclude that redistributing towards a suitable data partition for the QR factorization leads to significant improvement, compared to not redistributing the data as with the *owner computes* strategy. Initial distributions in *SkewedSet* are a good example where redistributing data is essential. Sometimes, like in *ChunkSet*, redistribution strategies involve a bigger amount of communications during the QR factorization but lead to a better load balancing across processors, which is enough to be profitable in the end.

	Vol. of comm. in the redist. phase				Nb. of steps in the redist. phase				Vol. of comm. in QR fact.		Total completion time				% Improv. / owner		
n	owner	can	vol	step	owner	can	vol	step	owner	can-vol-step	owner	can	vol	step	can	vol	step
16	0	249	119	119	0	37	32	32	1,973	831	3.34	1.94	2.02	1.89	41.9	39.5	43.4
34	0	1,119	538	538	0	157	149	149	15,118	5,720	25.22	9.06	8.14	8.48	64.1	67.7	66.4
52	0	2,616	1,257	1,257	0	378	353	353	49,707	16,669	78.11	26.75	23.07	22.51	65.8	70.5	71.2
70	0	4,739	2,286	2,286	0	680	638	638	114,127	37,942	182.96	53.44	50.11	52.32	70.8	72.7	71.4
88	0	7,492	3,615	3,615	0	1,092	1,019	1,019	219,757	69,951	344.92	100.87	94.61	95.14	70.8	72.6	72.4

(a) Results for *SkewedSet*

	Vol. of comm. in the redist. phase				Nb. of steps in the redist. phase				Vol. of comm. in QR fact.		Completion time				% Improv. / owner		
n	owner	can	vol	step	owner	can	vol	step	owner	can-vol-step	owner	can	vol	step	can	vol	step
16	0	240	205	233	0	34	31	31	1140	831	2.63	1.92	1.89	1.89	27.0	28.1	28.1
34	0	1,087	1,004	1,072	0	153	142	142	6,380	5,720	11.05	8.77	8.23	8.61	20.6	25.5	22.1
52	0	2,526	2,425	2,518	0	349	338	338	19,295	16,669	30.22	26.12	22.41	22.31	13.6	25.8	26.2
70	0	4,598	4,459	4,575	0	681	660	659	29,606	37,942	61.16	53.41	52.29	58.21	12.7	14.5	4.8
88	0	7,271	7,129	7,242	0	963	951	951	45,311	69,951	114.69	100.88	96.70	99.36	12.0	15.7	13.4

(b) Results for *ChunkSet*

Table 1: Results for the different initial distributions.

7 Conclusion

In this paper, we have studied the problem of finding the best data redistribution, given a target data partition. We have used two cost metrics, the total volume of communications and the number of parallel redistribution steps. We have provided algorithms computing the optimal solution for both metrics, and shown through simulations that they achieve significant gain over redistributing to an arbitrary fixed distribution. We have also proved that finding the optimal data partition that minimizes the completion time of the redistribution followed by a 1D-stencil kernel is NP-complete. Altogether, these results lay the theoretical foundations of the data partition problem on modern computers.

Admittedly, the platform model used in this paper will only be a coarse approximation of actual parallel performance, because state-of-the-art runtimes use intensive prefetching and overlap communications with computations. Therefore, experimental validation of the algorithms on a multicore cluster have been presented for a 1D-stencil kernel and a dense linear algebra routine. The new redistribution strategies presented in this paper lead to better performance in all cases, and the improvement is significant when the initial data distribution is not well-suited for the computational kernel.

Future work will be devoted to further investigating the Earth Science application. We have restricted to redistributing data towards the canonical 2D block-cyclic partition, but more experiments are needed to determine the best partition, given the initial distributions that typically arise for this application.

Acknowledgments.

The authors are with Université de Lyon, France, and with University of Tennessee, Knoxville. Y. Robert is with the Institut Universitaire de France. This work was supported in part by the France ANR *RESCUE* and *SOLHAR* (ANR-13-MONU-0007) projects, and by the USA Department Of Energy award DE-SC0010682. A preliminary and much shorter version of this work appears in the proceedings of ISPDC [33]. We would like to thank the reviewers for their comments and suggestions, which greatly helped improve the final version of the paper.

References

[1] L. Hollermann, T. S. Hsu, D. R. Lopez, and K. Vertanen, "Scheduling problems in a practical allocation model," *J. Combinatorial Optimization*, vol. 1, no. 2, pp. 129–149, 1997.

[2] T. S. Hsu, J. C. Lee, D. R. Lopez, and W. A. Royce, "Task allocation on a network of processors," *IEEE Trans. Computers*, vol. 49, no. 12, pp. 1339–1353, 2000.

[3] M. G. Norman and P. Thanisch, "Models of machines and computation for mapping in multicomputers," *ACM Computing Surveys*, vol. 25, no. 3, pp. 103–117, 1993.

[4] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, *Scheduling and load balancing in parallel and distributed systems*. IEEE CS Press, 1995.

[5] P. Chrétienne, E. G. Coffman Jr., J. K. Lenstra, and Z. Liu, Eds., *Scheduling Theory and its Applications*. John Wiley and Sons, 1995.

[6] H. El-Rewini, H. H. Ali, and T. G. Lewis, "Task scheduling in multiprocessing systems," *Computer*, vol. 28, no. 12, pp. 27–37, 1995.

[7] O. Beaumont, V. Boudet, and Y. Robert, "A realistic model and an efficient heuristic for scheduling with heterogeneous processors," in *HCW'2002, the 11th Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2002.

[8] P. Bhat, C. Raghavendra, and V. Prasanna, "Efficient collective communication in distributed heterogeneous systems," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 251–263, 2003.

[9] P. Rivera-Vega, R. Varadarajan, and S. Navathe, "Scheduling data redistribution in distributed databases," in *Proc. Sixth Int. Conf. Data Engineering*, 1990, pp. 166–173.

[10] Y.-A. Kim, "Data migration to minimize the total completion time," *J. Algorithms*, vol. 55, no. 1, pp. 42–57, 2005.

[11] E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. LaPaugh, "Scheduling file transfers," *SIAM J. Comput.*, vol. 14, pp. 744–780, 1985.

[12] E. Anderson, J. Hall, J. Hartline, M. Hobbes, A. Karlin, J. Saia, R. Swaminathan, and J. Wilkes, "Algorithms for data migration," *Algorithmica*, vol. 57, no. 2, pp. 349–380, 2010.

[13] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. S. Jr., and M. E. Zosel, *The High Performance Fortran Handbook*. The MIT Press, 1994.

[14] J. J. Dongarra and D. W. Walker, "Software libraries for linear algebra computations on high performance computers," *SIAM Review*, vol. 37, no. 2, pp. 151–180, 1995.

[15] E. T. Kalns and L. M. Ni, "Processor mapping techniques towards efficient data redistribution," *IEEE Trans. Parallel Distributed Systems*, vol. 6, no. 12, pp. 1234–1247, 1995.

[16] D. W. Walker and S. W. Otto, "Redistribution of block-cyclic data distributions using MPI," *Concurrency: Practice and Experience*, vol. 8, no. 9, pp. 707–728, 1996.

[17] L. Wang, J. M. Stichnoth, and S. Chatterjee, "Runtime performance of parallel array assignment: an empirical study," in *1996 ACM/IEEE Supercomputing Conference*, 1996.

[18] R. Thakur, A. Choudhary, and G. Fox, "Runtime array redistribution in hpf programs," in *Scalable High-Performance Computing Conference, 1994., Proceedings of the*, May 1994, pp. 309–316.

[19] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert, "Scheduling block-cyclic array redistribution," *IEEE Trans. Parallel Distributed Systems*, vol. 9, no. 2, pp. 192–205, 1998.

[20] M. Guo and Y. Pan, "Improving communication scheduling for array redistribution," *J. Parallel Distrib. Comput.*, vol. 65, no. 5, 2005.

[21] L. Prylli and B. Tourancheau, "Efficient block-cyclic data redistribution," in *EuroPar'96*, ser. Lectures Notes in Computer Science, vol. 1123. Springer Verlag, 1996, pp. 155–164.

[22] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, ser. Algorithms and Combinatorics. Springer-Verlag, 2003, vol. 24.

[23] J. E. Hopcroft and R. M. Karp, "An n^5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[24] G. Smith, *Numerical Solutions of Partial Differential Equations: Finite Difference Methods*. Clarendon Press, Oxford, 1985.

[25] M. R. Garey and D. S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[26] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra, "DAGuE: A generic distributed DAG engine for high performance computing," *Parallel Computing*, vol. 38, no. 1, pp. 37–51, 2012.

[27] ——, "DAGuE: A generic distributed DAG engine for high performance computing," in *16th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'11)*, 2011.

[28] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, "A class of parallel tiled linear algebra algorithms for multicore architectures," *Parallel Computing*, vol. 35, pp. 38–53, 2009.

[29] G. Quintana-Ortí, E. S. Quintana-Ortí, R. A. van de Geijn, F. G. Van Zee, and E. Chan, "Programming matrix algorithms-by-blocks for thread-level parallelism," *ACM Trans. Math. Softw.*, vol. 36, no. 3, pp. 1–26, 2009.

[30] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. Whaley, "ScaLAPACK: a portable linear algebra library for distributed memory computers — design issues and performance," *Computer Physics Communications*, vol. 97, no. 1–2, p. 1–15, 1996.

[31] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, A. Haidar, T. Herault, J. Kurzak, J. Langou, P. Lemarinier, H. Ltaief, P. Luszczek, A. YarKhan, and J. Dongarra, "Flexible development of dense linear algebra algorithms on massively parallel architectures with DPLASMA," in *12th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC'11)*, 2011.

[32] M. Stonebraker, J. Duggan, L. Battle, and O. Papaemmanouil, "SciDB DBMS research at M.I.T," *IEEE Data Eng. Bull.*, vol. 36, no. 4, pp. 21–30, 2013.

[33] T. Herault, J. Herrmann, L. Marchal, and Y. Robert, "Determining the optimal redistribution for a given data partition," in *Parallel and Distributed Computing (ISPDC), 2014 IEEE 13th International Symposium on*. IEEE, 2014, pp. 95–102.